On the suitability of black-box performance monitoring for SLA-driven cloud provisioning scenarios

Arnaud Schoonjans, Dimitri Van Landuyt, Bert Lagaisse, Wouter Joosen
iMinds-DistriNet, KU Leuven
3001 Leuven, Belgium
firstname.lastname@cs.kuleuven.be

ABSTRACT

In recent years, cloud computing has become an increasingly important software delivery paradigm, mainly for reasons of increased scalability. The scalability benefits are accomplished by the capability of autonomously and elastically scaling up or down so that customer preferences (SLAs) can be accommodated. For this, performance monitoring is a prerequisite. Distinction is made between white-box and black-box monitoring techniques: the former involves collecting information about the monitored component by looking at its internals, while the latter only involves observing the components interfaces.

In practice, cloud provisioning is commonly based on white-box monitoring. These techniques are costly to develop, since technologies (and providers) offer their own white-box inspection APIs and are costly to integrate (e.g., in a multi-cloud setup involving different providers). In addition it is not always possible to apply this performance monitoring technique when dealing with third-party components or services.

In this paper, we investigate whether typical SLA-driven cloud provisioning scenarios can be supported when relying exclusively on black-box performance monitoring techniques. We perform an experiment in which we apply both white-box (e.g., CPU usage, load, etc.) and black-box instrumentation (e.g., latency of operations, amount of failed operations, etc.) in a realistic case study, and we discover clear correlations between some of the obtained white-box and black-box measurements. As such, we show that black-box performance monitoring techniques can be used to support such provisioning scenarios.

Categories and Subject Descriptors

K.6 [**MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS**]: System Management

General Terms

Performance, Measurement

ARM 2015, December 07-11, 2015, Vancouver, BC, Canada

DOI: http://dx.doi.org/10.1145/2834965.2834971.

1. INTRODUCTION

Cloud computing is increasingly becoming a strategic choice in the development of applications and services, because of increased scalability, the pay-as-you-use billing model, etc. For this reason, cloud providers themselves start using components, offered by third-party cloud providers, as building blocks for their applications. This leads to heterogeneous multi-cloud applications, where only the necessary amount of resources are acquired from third-party cloud providers to run an application at customer-defined service levels. Supporting such SLA-driven cloud provisioning scenarios requires a monitoring and provisioning control loop [8] that involves observing the offered service levels and triggers reconfiguration, i.e. up- or down-scaling the cloud application. In this paper, we focus on the required monitoring capabilities to realize the above-mentioned cloud provisioning control loop.

In component-based software engineering [19], a component is defined as: *"a non-trivial, nearly independent and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture."* Interaction with a component from the outside happens through a set of well-defined interfaces. Monitoring approaches in such a component-based context range from pure white-box monitoring to pure black-box monitoring. A pure white-box (WB) monitoring technique collects performance measurements about a certain component using information gathered inside that component [14]. A pure black-box (BB) performance monitoring technique only needs to know the interface of the component to perform the monitoring activity [14]. The left side column of Table 1 lists a few categories of black-box and white-box metrics. Note that black-box performance monitoring requires the performance metrics to be collected using another component in the distributed system that interacts with the service interface of the monitored component.

Monitoring the performance of third-party services (*Monitoring* phase) to support these SLA-driven provisioning scenarios, is often difficult. In most cases, cloud providers only provide access to the interface of the offered component (e.g., a REST API) without any information about the underlying infrastructure or the internal state of the component. A similar issue arises when the source code of a third-party component is not available.

Several trade-offs are involved in choosing between white-box and black-box monitoring. Pure white-box monitoring gathers most accurate information, but lead towards heavyweight, complex components that are hard to integrate, be-

cause of the heterogeneity of all different monitoring API's. An example of white-box monitoring in a cloud environment is given by Google cloud monitoring [15]. This monitoring tool provides plugins to monitor for instance an Elasticsearch database using the Elasticsearch build-in `Stats API` [13] (white-box monitoring API). Pure black-box monitoring is promising as the monitoring support is implemented in middleware and leads to more lightweight components. The cost of white-box instrumentation in our experiments turned out to be 6482 lines of code (2%), while only 30 lines of code were required for black-box instrumentation.

We identified the following three classes of scenarios where black-box performance monitoring is either promising or the only viable option: (i) a SaaS provider integrates different third-party components to one SaaS application and wants to monitor the performance of these integrated components, (ii) a SaaS provider uses the services of another SaaS provider and wants to monitor the performance of these external services and (iii) a customer wants to monitor a SaaS application to verify whether the service levels, guaranteed by the SaaS provider, correspond to the ones observed by the customer.

The above-mentioned scenarios clearly indicate the advantages of applying black-box performance monitoring. Of course, the collected black-box metrics need to reflect the current health state of the application sufficiently accurate for black-box performance monitoring to be useful afterall. This paper investigates whether *it is possible to support SLA-driven cloud provisioning scenarios, while relying only on black-box performance metrics.*

We present an experiment where both black-box and white-box performance monitoring are applied for monitoring the performance of a client interacting with an ElasticSearch database. The results show a clear correlation between the observed black-box and white-box measurements. This illustrates that black-box performance monitoring can be a valid alternative to white-box performance monitoring to support SLA-driven cloud provisioning scenarios. This paper does not present a general performance monitoring paradigm. Instead, we claim that black-box performance monitoring provides sufficient information about the achieved SLAs to be useful in SLA driven cloud provisioning scenarios, while the implementation-time and runtime overhead is rather limited.

The remainder of this paper is structured as follows. Section 2 motivates the problem and hypothesis of this paper in more detail. The concrete experiment setup for the verification of the hypothesis and the associated results are presented in Sections 3 and 4. Section 5 gives an overview of the related work and Section 6 concludes this paper.

2. MOTIVATION: SLA PROVISIONING IN CLOUDS

The motivation for this paper is based on our findings in a collaborative research project with industry [12], which involves a multi-cloud Log-Management-as-a-Service (LMaaS) application. More specifically, we encountered a number of cloud provisioning-related scenarios in which the SaaS provider does not have total control over a certain part of the application, and in which pure white-box monitoring either is undesirable or unfeasible:

1. The LMaaS application relies on an ElasticSearch installation (both privately and publicly hosted, in a multi-cloud setup). When integrating the private deployment of Elasticsearch, the SaaS offering must integrate with the white-box inspection APIs of ElasticSearch (the `stats API`), which involves substantial implementation overhead, and increases the risk of vendor and technology lock-in.

2. The same applies when the ElasticSearch services are hosted by a third party provider (Data Storage as a Service) and integrated into the SaaS application. In addition, there are no guarantees about access to inspection interfaces and there is the additional risk of provider lock-in (for example, when the provider offers his own inspection interfaces).

3. Finally, black-box performance monitoring can be applied by a customer to verify whether the level of service delivered to him or her is in accordance with the SLAs, guaranteed by the cloud provider. This is true for customers of the LMaaS system, but also in the context of the second scenario in which the LMaaS provider himself is a customer of the ElasticSearch service provider.

Although these scenarios provide promising arguments in favor of black-box monitoring, the key question remains whether black-box metrics can entirely replace white-box metrics in the context of such cloud provisioning scenarios.

The experiment presented in this paper tests our hypothesis that **it is indeed possible to support SLA-driven cloud provisioning scenarios, while relying only on black-box performance metrics**.

3. EXPERIMENT SETUP

Figure 1 shows the deployment of the experiment[1]. Three nodes are involved: the client node, the ElasticSearch node and the measurements collector node. The client node contains an extended version of the YCSB benchmarking tool [9], used to evaluate the performance of different NoSQL storage systems. The YCSB tool supports measuring black-box metrics about the query executions at the middleware level. The Elasticsearch node runs a standard installation of the Elasticsearch database. As the figure indicates, two interfaces are used to interact with the ElasticSearch service component. The first one is the query interface, which enables a client to issue queries on the database. The second one is the ElasticSearch `stats API` [13], that allows a client to retrieve white-box performance measurements. Finally, the measurements collector node provides storage and aggregation support for both white-box and black-box metrics collected during the experiment.

During the experiment, the extended YCSB tool simulates a specific amount of users executing queries on the database. The type of query executed by the YCSB tool in this specific setup is a read-modify-write operation. All inserted data is randomly generated. The `service request wrapper` within the client component is the middleware collecting black-box metrics about the query executions. The collected measurements are sent to the `measurements collector` component deployed on a third node.

[1]The code used for this experiment is available at: `http://people.cs.kuleuven.be/~arnaud.schoonjans/arm_2015/`

The YCSB benchmarking tool gradually increases the load on the ElasticSearch service component, by increasing the query request rate. The load applied during the experiment is situated between the point where almost no load is applied to the database system and goes up to right before the overload point. The experiment is not performed under overload conditions for two reasons. First of all, it is very likely that black-box and white-box metrics correlate differently under non-overload conditions compared to overload conditions. It might for example be the case that a black-box and a white-box metric correlate in a linear way when not overloaded while an exponential relation is found in the overload part of the load range. Second, the monitored system should never go past the overload point since the target of an autonomous system is to detect hight load and take action before the overload point is reached (and thus SLA violation occurs).

The `measurements collector` component retrieves black-box measurements from the `service request wrapper` component. At the same time, it requests white-box metrics from the Elasticsearch database node using the Elasticsearch `stats API`.

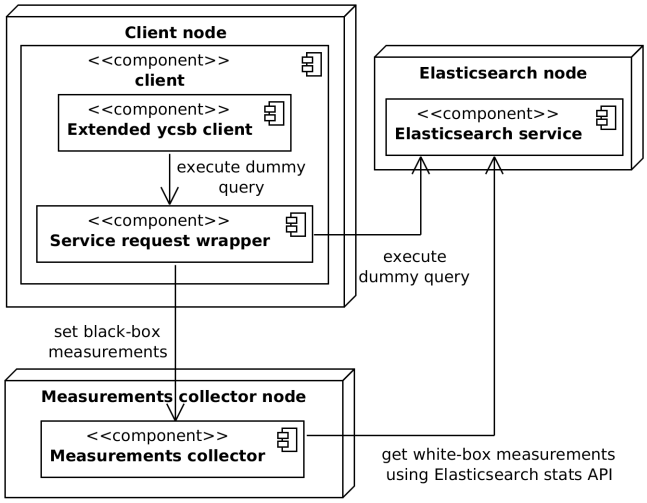


Figure 1: Experiment setup (deployment) used to collect white-box and black-box measurements from the interaction with an ElasticSearch database

The black-box and white-box metrics applied during the experiment are presented in Table 1. The left-hand side column of this table divides the metrics into several categories, while the right-hand side column lists the actual metrics[2].

To quantify the correlation between these white-box and black-box performance metrics, the Spearman rank correlation coefficient [10] is calculated for every combination of white-box and black-box metric highlighted in Table 1. This is an interesting coefficient for our evaluation, because we want to quantify the correlation between certain white-box and black-box metrics, while the type of correlation is not important (linear, exponential, etc.).

[2]The white-box metrics mentioned in this table are a subset of all white-box metrics offered by the ElasticSearch `stats API`. The selection of this subset is based on their relation with performance.

White-box metrics	
Load	`load1`, `load5`, **load15**
CPU usage	**cpuUserInMillis**, `cpuSysInMillis`, `cpuTotalInMillis`
Memory usage	**memResidentInBytes**, `memTotalVirtualInBytes`
Garbage collection	`gcTimeYoungInMillis`, **gcTimeOldInMillis**
JVM Threads	**amountOfThreadsInJvm**
Open files	**amountOfOpenFileDescriptors**
Connections	**amountOfOpenConnections**
Black-box metrics	
Latency	**averageLatencyInMillis**, **90percentileLatencyInMillis**, **peakLatencyInMillis**

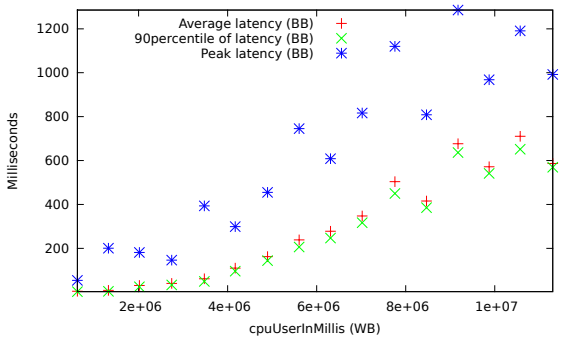
Table 1: An overview of the white-box and black-box metrics measured during the experiment. The metrics selected for our correlation analysis are shown in bold.

4. RESULTS

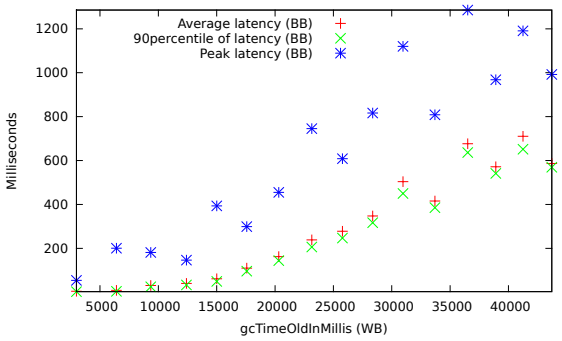
This section presents the results from the experiment described in section 3. The results show that all metrics in the same category (eg. `load1 (WB)`, `load5 (WB)` and `load15 (WB)`) are correlated, because they for example present the same metric at a different resolution. The remainder of this section describes the results for the following metrics as a representative of their category: `load15 (WB)` for load, `gcTimeOldInMillis (WB)` for garbage collection time, `cpuUserInMillis (WB)` for the CPU usage and the `memResidentInBytes (WB)` for the memory usage.

The Spearman rank correlation coefficient between each of the black-box and latency-related white-box metrics are shown in Table 2. The results indicate a strong correlation between the black-box metrics `cpuUserInMillis (WB)`, `memResidentInBytes (WB)`, `gcTimeOldInMillis (WB)` and `amountOfOpenFileDescriptors (WB)` and the latency relation white-box metrics. The metrics `load15 (WB)` and `amountOfOpenConnections (WB)` are less strongly correlated with an absolute correlation coefficient between 0.77 and 0.82. Finally, the `amountOfThreadsInJvm (WB)` metric has the worst correlation with each of the latency related white-box metrics. The Spearman correlation coefficients have values between 0.57 and 0.58.

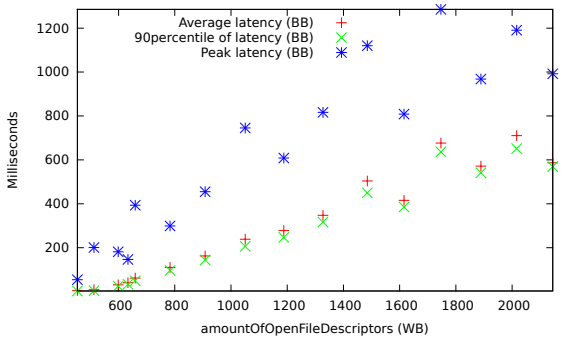
To further illustrate the nature of these correlations, the scatterplots in Figure 2 illustrate the observed relations between different white-box and black-box metrics. For each plot, the X-axis contains one of the white-box metrics from Table 1, the Y-axis on the other hand gives the latency in milliseconds for each of the latency-related black-box metrics (`averageLatencyInMillis (BB)`, `90PercentileLatencyInMillis (BB)` and `peakLatencyInMillis (BB)`). Figures 2a, 2b and 2c illustrate the linear correlation between the metrics `cpuUserInMillis (WB)`, `gcTimeOldInMillis (WB)` and `amountOfOpenFileDescriptors (WB)` and the latency-related metrics (BB), while an exponential correlation is suggested for the `memResidentInbytes (WB)`, `amountOfThreadsInJvm (WB)` and the `load15 (WB)` metric in Figures 2d, 2f and 2g. For the `amountOfOpenConnections (WB)` metric, given in Figure 2e, a weak linear correlation can be observed.



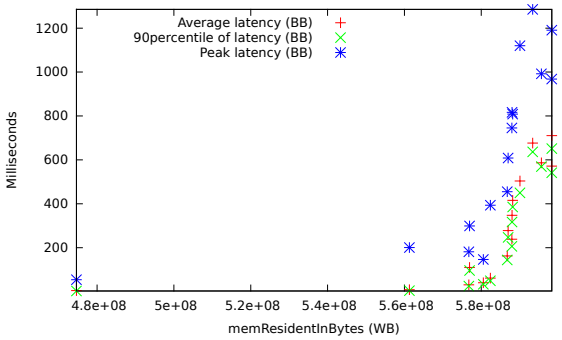
(a) `cpuUserInMillis (WB)` vs. latency metrics (BB)



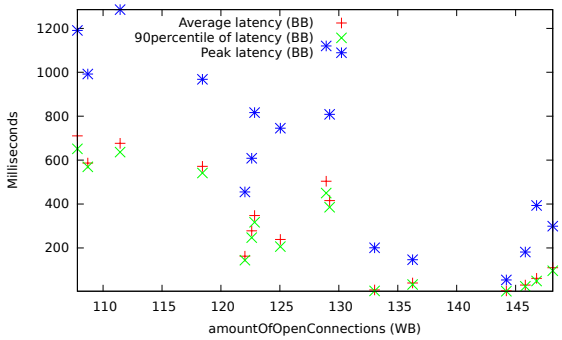
(b) `gcTimeOldInMillis (WB)` vs. latency metrics (BB)



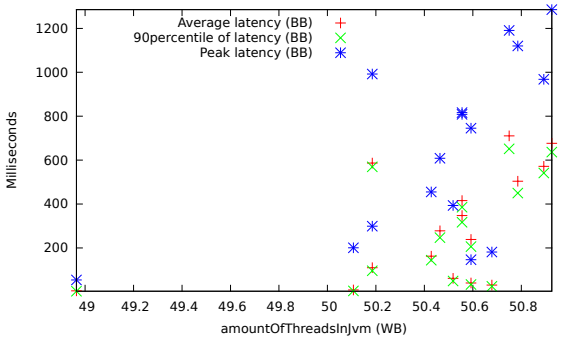
(c) `amountOfOpenFileDescriptors (WB)` vs. latency metrics (BB)



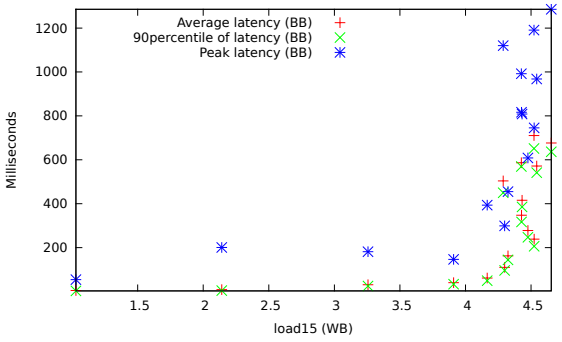
(d) `memResidentInBytes (WB)` vs. latency metrics (BB)



(e) `amountOfOpenConnections (WB)` vs. latency metrics (BB)



(f) `amountOfThreadsInJvm (WB)` vs. latency metrics (BB)



(g) `load15 (WB)` vs. latency metrics (BB)

Figure 2: Scatterplots illustrating the correlation between white-box and black-box metrics. The X-axis plots white-box measurements against three latency-related black-box measurements on the Y-axis.

Black-box metric	Spearman correlation coefficient		
	averageLatencyInMillis	**peakLatencyInMillis**	**90percentileLatencyInMillis**
load15	0.82	0.77	0.82
cpuUserInMillis	0.98	0.92	0.98
memResidentInBytes	0.97	0.94	0.97
gcTimeOldInMillis	0.98	0.92	0.98
amountOfThreadsInJvm	0.57	0.58	0.57
amountOfOpenFileDescriptors	0.98	0.92	0.98
amountOfOpenConnections	-0.81	-0.78	-0.81

Table 2: Spearman correlation coefficient between black-box metrics and the latency-related white-box metrics (`averageLatencyInMillis (BB)`, `90percentileLatencyInMillis (BB)`, `peakLatencyInMillis (BB)`)

Table 3 summarizes the main results of our experiment, assuming that a correlation exists when the absolute value of the Spearman correlation coefficient is larger than 0.9.

Discussion.

The experiment results show a clear correlation between some of the black-box and some of the white-box metrics. As a consequence, we can now conclude that, for typical cloud provisioning SLAs which involve these highly-correlated white-box metrics such as CPU or memory usage, these scenarios can be accomplished by relying exclusively on the black-box metrics.

The downside of black-box performance monitoring is that black-box metrics in most cases only detect the existence of a performance problem while they cannot identify the source of the problem. For example, a black-box performance monitoring technique will detect that the latencies of requests to the monitored component increase, without any knowledge about the source of the problem. A white-box performance monitoring technique on the other hand will detect that the JVM, running the monitored component, is short on memory causing a high garbage collection overhead. The reconfiguration action is obvious in the white-box monitoring case, while this is not obvious when performing black-box monitoring.

Note that exact knowledge of the cause of a performance issue (for which white-box monitoring is a necessity) is not always desirable in the context of cloud computing. Indeed, the response upon detection of SLA violation never involves searching for and addressing the internal root cause of the problem but dynamically and elastically up- or downscaling the cloud service (i.e. ∼re-provisioning).

5. RELATED WORK

Cloud systems by definition involve elastic provisioning [22]. In many cases, these provisioning scenarios are driven by SLAs that stipulate performance requirements, availability requirements, etc. This requires an SLA monitoring and provisioning control loop [5, 8, 11]. Such a control loop consists of the following phases [7, 20]: (i) monitoring the performance of a certain component (monitoring phase), (ii) comparing the observed service levels to the desired service levels written down in an SLA policy (analyse phase) and (iii) performing automatic reconfiguration in case of SLA violations (plan and execute phase). The goal is to automatically up- and down-scale components when required. Huebscher et al. [17] highlight the different design decisions in an autonomous system. This includes decisions involving the selection of adequate metrics, the monitoring strategy and frequency.

The importance of monitoring in the context of maintaining a component-based distributed system has been pointed out by many. Mari et al. [19] show that the maintenance cost of components becomes increasingly important compared to the actual development cost, since distributed systems are increasingly composed of third-party software components. Other related work focuses on the advantages and the need for black-box performance monitoring in specific circumstances [2, 4, 6]. Canfora el al. [6] for example, highlight that white-box performance monitoring becomes difficult when access to a certain component is limited to its interface. Aversa et al. [2] on the other hand, point to the conflict of interest that occurs when monitoring features are offered by the service provider himself, which provides clear motivation for the use of black-box performance monitoring by the customer to verify whether the delivered service level is in line with the agreed-upon SLA.

Next to that, a significant amount of research has been performed on black-box performance monitoring within a distributed system. Several tools have been developed to perform black-box performance monitoring, operating from different perspectives and within different contexts. The monitoring support can be integrated into the hypervisor [25–27], into the kernel of the OS [23], into the component itself by putting a wrapper around the components interface [16, 24], into the network by monitoring all messages passing by [1], etc. Our approach is similar in the sense that we want to reduce implementation and integration overhead of monitoring support, but our approach assumes that we do not have any access to the monitored system or infrastructure.

Some research has been done in the development of tools that use black-box performance monitoring during the deployment phase [3, 18]. Kuperberg et al. [18], for example, estimate the performance of a black-box component. This is accomplished by analysing the bytecode of the component and comparing these values to the execution time of certain bytecode on a specific deployment platform. As a consequence, the performance of an application can be predicted without specific knowledge of the component source code.

Finally, Mdhaffar et al. [21] propose a combination of black-box and white-box monitoring by instrumenting the client and the server application using aspect oriented programming. As such, they perform end-to-end monitoring, while the combination of black-box and white-box monitoring allows the monitoring infrastructure to make a distinction between execution time (server side computations) and communication time (network overhead). This shows that a combination of both monitoring techniques is also valuable.

Black-box metrics	white-box metrics	Type of correlation
Latency	CPU usage, garbage collection, open files	Linear
Latency	memory usage	Exponential
Latency	JVM threads, load, connections	No strong correlation exists

Table 3: Overview: type of correlation between white-box and black-box metric categories

6. CONCLUSION

This paper provides empirical evidence that it is indeed possible, for the case study presented in this paper, to collect sufficient information about the load on a certain component only using black-box monitoring techniques. We can conclude that black-box performance monitoring can be valuable for component developers and integrators. Developers are not required to instrument components with heavyweight, costly white-box performance monitoring support. Component integrators on the other hand are not required to care about the monitoring support provided by a third-party component or third-party service provider, because black-box performance monitoring is always feasible. This way, black-box performance monitoring can be a good addition to white-box performance monitoring in case white-box monitoring is applicable or a good alternative in case white-box performance monitoring is not an option.

In future work, we will extend and repeat our experiment considering for example conditions of network failure, network saturation, etc. As such, we will try to find out whether black-box performance monitoring is able to notice these network-related problems.

Acknowledgment

This research is partially funded by the Research Fund KU Leuven (project GOA/14/003 - ADDIS) and the iMinds DMS2 project, which is co-funded by iMinds (Interdisciplinary institute for Technology), a research institute founded by the Flemish Government. Companies and organizations involved in the project are Agfa Healthcare, Luciad, UP-nxt, and Verizon Terremark, with project support of IWT (government agency for Innovation by Science and Technology).

7. REFERENCES

[1] Muthitacharoen et al. Aguilera, Patrick. Performance debugging for distributed systems of black boxes. *SIGOPS Oper. Syst. Rev.*, 2003.

[2] Venticinque et al. Aversa, Tasquier. Agents based monitoring of heterogeneous cloud infrastructures. 2013.

[3] Polini et al. Bertolino, Muccini. Architectural verification of black-box component-based systems. Springer, 2007.

[4] Jan Bosch. Adapting object-oriented components. Springer-Verlag, 1998.

[5] Di Marzo et al. Brun, Serugendo. Engineering self-adaptive systems through feedback loops. Springer, 2009.

[6] G. Canfora and M. Di Penta. Testing services and service-centric systems: challenges and opportunities. *IT Professional*, 2006.

[7] Luca Casalicchio, Silvestri. Architectures for autonomic service management in cloud-based systems. IEEE, 2011.

[8] Autonomic Computing. An architectural blueprint for autonomic computing. *IBM Publication*, 2003.

[9] Tam et al. Cooper, Silberstein. Benchmarking cloud serving systems with YCSB. *Proceedings of the 1st ACM symposium on Cloud computing SoCC 10*, 2010.

[10] Dale Corder, Foreman. *Nonparametric statistics: A step-by-step approach*. John Wiley & Sons, 2014.

[11] Müller et al. De Lemos, Giese. Software engineering for self-adaptive systems: A second research roadmap.

[12] DMS2. Decentralized Data Management and Migration for SaaS (iMinds ICON project).

[13] Elasticsearch. Elasticsearch Node Stats API. `https://www.elastic.co/guide/en/elasticsearch/reference/1.5/cluster-nodes-stats.html`, 2015.

[14] Shim et al. Gao, Zhu. Monitoring software components and component-based software. 2000.

[15] Google. Google cloud monitoring. `https://cloud.google.com/monitoring/agent/`, 2015.

[16] Dashofy Hendrickson and Taylor. An approach for tracing and understanding asynchronous architectures. 2003.

[17] Julie Huebscher, McCann. A survey of autonomic computing - degrees, models, and applications. 2008.

[18] Reussner et al. Kuperberg, Krogmann. Performance prediction for black-box components using reengineered parametric behaviour models. Springer, 2008.

[19] M. Mari and N. Eila. The impact of maintainability on component-based software systems. 2003.

[20] Emeakaroha et al. Maurer, Breskovic. Revealing the mape loop for the autonomic management of cloud infrastructures. IEEE, 2011.

[21] Juhnke et al. Mdhaffar, Ben Halima. Aop4csm: An aspect-oriented programming approach for cloud service monitoring. IEEE, 2011.

[22] Timothy Mell, Grance. The NIST definition of cloud computing. Technical report, NIST, 2011.

[23] Adrian Mos and John Murphy. A framework for performance monitoring, modelling and prediction of component oriented distributed systems. ACM, 2002.

[24] Mogul et al. Reynolds, Wiener. Wap5: black-box performance debugging for wide-area systems. ACM, 2006.

[25] Bala et al. Suneja, Isci. Non-intrusive, out-of-band and out-of-the-box systems monitoring in the cloud. ACM, 2014.

[26] Zhang et al. Tak, Tang. vpath: Precise discovery of request processing paths from black-box observations of thread and network activities. USENIX, 2009.

[27] Arun Venkataramani et al. Timothy Wood, Prashant Shenoy. Sandpiper: Black-box and gray-box resource management for virtual machines. *Computer Networks*, 2009. Virtualized Data Centers.